课程回顾

- 动态规划概述
  - 算法步骤：刻画一个最优解的结构特征、递归地定义最优解的值、计算最优解的值，通常采用自底向上的方法、利用计算出的信息构造一个最优解
  - 实现方法：带备忘的自顶向下法、自底向上法
- 动态规划问题：钢条切割

钢条切割问题

- Serling公司购买长钢条将其切割为短钢条出售
- 切割工序本身没有成本
- 出售长度为i英寸的钢条价格为p_i (i=1, 2, ..., 10)美元

长度i	1	2	3	4	5	6	7	8	9	10
价格p_i	1	5	8	9	10	17	17	20	24	30

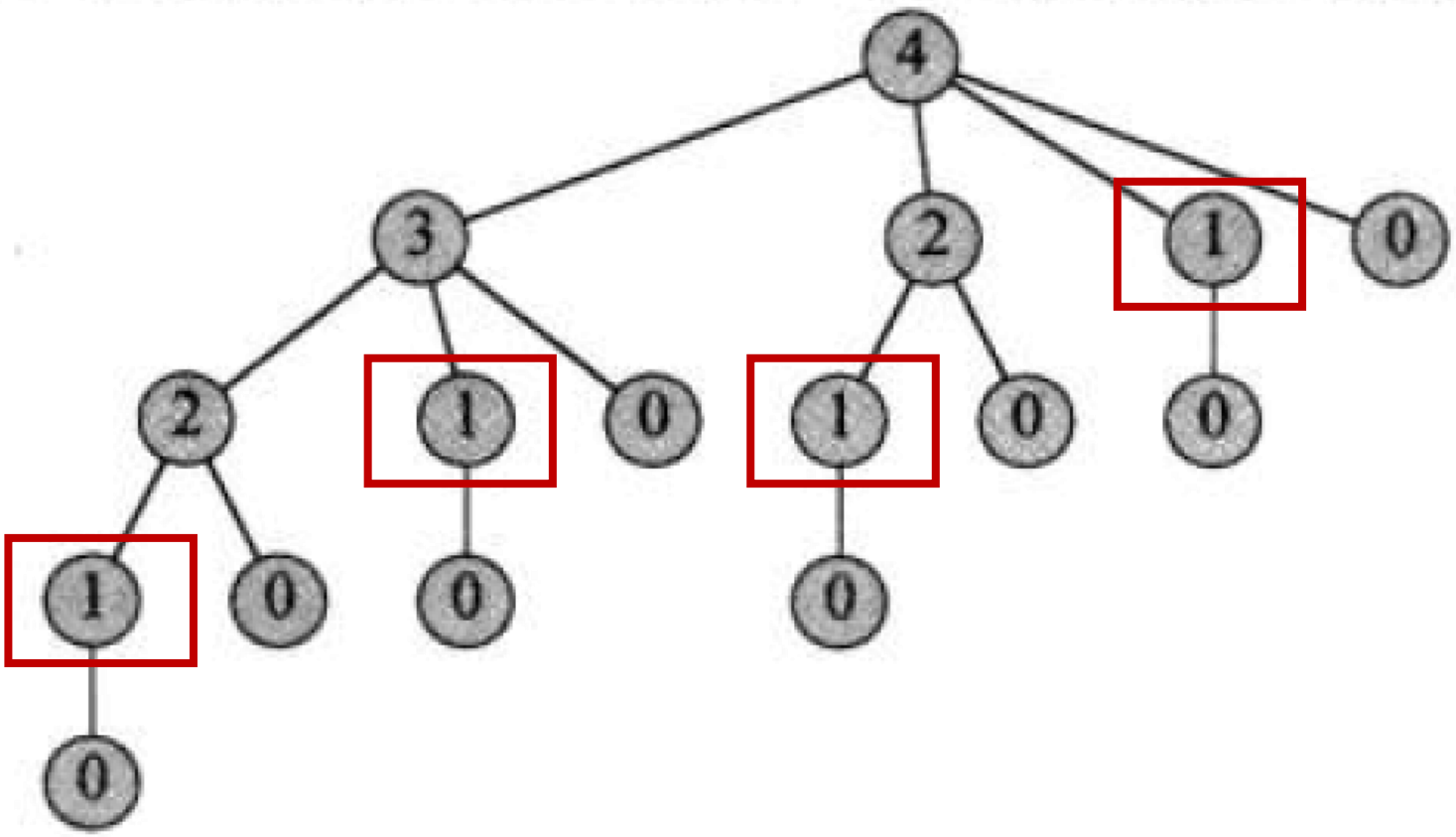
- 钢条切割问题定义：给定一段长度为n英寸的钢条和一个价格表p_i (i=1, 2, ..., n)，求切割钢条方案，使得销售收益r_n最大

钢条切割问题 (续)

- 最优子结构：问题的最优解由相关子问题的最优解组合而成，而这些子问题可以独立求解
- 简化求解：从左边切割下长度为i的一段不再继续切割，只对右边剩下的n-i段继续切割（递归求解）：$r_n = \max_{1 \le i \le n} (p_i + r_{n-i})$
- 自顶向下递归：

```
CUT_ROD(p, n)
1  if n = 0
2      return 0
3  q ← -∞
4  for i ← 1 to n do
5      q ← max(q, p[i] + CUT_ROD(p, n-i))
6  return q
```

钢条切割问题 (续)



钢条切割问题 (续)

```
CUT_ROD(p, n)
1  if n = 0
2      return 0
3  q ← -∞
4  for i ← 1 to n do
5      q ← max(q, p[i] + CUT_ROD(p, n-i))
6  return q
```

■时间复杂度分析：

$$T(n) = 1 + \sum_{i=1}^{n} T(n-i)$$

$$= 1 + T(n-1) + T(n-2) + \ldots + T(0)$$

$$T(n-1) = 1 + T(n-2) + T(n-3) + \ldots + T(0)$$

$$T(n) - T(n-1) = T(n-1) \Rightarrow T(n) = 2T(n-1)$$

$$\Rightarrow T(n) = 2^n$$

指数时间复杂度！

钢条切割问题 (续)

■带备忘的自顶向下法：

```
MEMOIZED_CUT_ROD(p, n)
1  let r[0..n] be a new array
2  for i ← 0 to n do
3      r[i] ← -∞
4  return MEMOIZED_CUT_ROD_AUX(p, n, r)
```

```
MEMOIZED_CUT_ROD_AUX(p, n, r)
1  if r[n] ≥ 0
2      return r[n]
3  if n = 0
4      q ← 0
5  else q ← -∞
6      for i ← 1 to n
7          q ← max(q, p[i] + MEMOIZED_CUT_ROD_AUX(p, n-i, r))
8  r[n] ← q
9  return q
```

钢条切割问题 (续)

■自底向上法：

```
BOTTOM_UP_CUT_ROD(p, n)
1  let r[0..n] be a new array
2  r[0] ← 0
3  for j ← 1 to n do
4      q ← -∞
5      for i ← 1 to j do
6          q ← max(q, p[i] + r[j-i])
7      r[j] ← q
8  return r[n]
```

➢依次求解规模为$j = 0, 1, ..., n$的子问题

钢条切割问题 (续)

- 重构解：输出最大收益及切割方案
- 对长度为j的钢条计算最大收益r_j及第一段钢条切割长度s_j

```
EXTENDED_BOTTOM_UP_CUT_ROD(p, n)
1   let r[0..n] be a new array
2   r[0] ← 0
3   for j ← 1 to n do
4       q ← -∞
5       for i ← 1 to j do
6           if q < p[i] + r[j-i]
7               q ← p[i] + r[j-i]
8               s[j] ← i
9           r[j] ← q
10  return r and s
```

钢条切割问题 (续)

■输出最优切割方案

```
PRINT_CUT_ROD_SOLUTION(p, n)
1  (r, s) ← EXTENDED_BOTTOM_UP_CUT_ROD(p, n)
2  while n > 0 do
3      print s[n]
4      n ← n – s[n]
```

动态规划问题

- 钢条切割
- 矩阵链乘法的最优括号化
- 多边形的最佳三角剖分
- 最长公共子序列
- 最优二叉搜索树
- 0-1背包

矩阵链乘法

- 给定n个矩阵的序列（矩阵链）$<A_1, A_2, ..., A_n>$，计算乘积$A_1A_2 \cdots A_n$
- 计算矩阵链乘积可用括号来决定计算次序，每一个括号内的矩阵相乘调用标准的矩阵乘法
- 矩阵积的完全括号化
  - 它是单一矩阵
  - 或者是两个完全括号化的矩阵链的积

递归定义
计算次序无二义性

$(A_1(A_2(A_3A_4)))$　　$((A_1(A_2A_3))A_4)$

$(A_1((A_2A_3)A_4))$　　$(((A_1A_2)A_3)A_4)$

$((A_1A_2)(A_3A_4))$

矩阵链乘法 (续)

■不同的括号化方式产生不同的计算成本

```
MATRIX_MULTIPLY(A, B)
1  if A.columns ≠ B.rows
2     error "incompatible dimensions"
3  else let C be a new A.rows×B.columns matrix
4     for i ← 1 to A.rows do
5        for j ← 1 to B.columns do
6           c_ij ← 0
7           for k ← 1 to A.columns do
8              c_ij ← c_ij + a_ik·b_kj
9  return C
```

- 第8行执行次数：A.*rows*×B.*columns*×A.*columns*(或B.*rows*)
- 设A是$p\times q$矩阵、B是$q\times r$矩阵，则计算C=A·B共需pqr次标量乘法

矩阵链乘法 (续)

■不同的括号化方式产生不同的计算成本

- 例：以矩阵链<A_1, A_2, A_3>相乘为例，三个矩阵规模分别为10×100、100×5、5×50
- $(A_1A_2)A_3$：A_1A_2：10×100×5=5000, 得到10×5矩阵

  $(A_1A_2)A_3$：10×5×50=2500

  共计5000+2500=7500次标量乘法
- $A_1(A_2A_3)$：A_2A_3：100×5×50=25000, 得到100×50矩阵

  $A_1(A_2A_3)$：10×100×50=50000

  共计25000+50000=75000次标量乘法

相差10倍！

矩阵链乘法 (续)

■矩阵链乘法问题实质上是一个最优括号化问题：

- 给定n个矩阵的链$<A_1, A_2, ..., A_n>$，矩阵A_i的规模为$p_{i-1} \times p_i$ $(1 \le i \le n)$，求完全括号化方案，使得计算乘积$A_1 A_2 \cdots A_n$所需标量乘法次数最少
- 计算括号化方案数量：设$P(n)$表示一个n个矩阵的链中可选括号化方案数量，则穷举法产生数量：

$$P(n) = \begin{cases} 1, & n = 1, \\ \sum_{k=1}^{n-1} P(k)P(n-k), & n \ge 2. \end{cases}$$

Catalan数，指数阶：$\Omega(4^n/n^{1.5})$，不如直接求解矩阵乘积！

矩阵链乘法 (续)

■应用动态规划方法可获得多项式时间求解方法

1. 刻画一个最优解的结构特征
2. 递归地定义最优解的值
3. 计算最优解的值，通常采用自底向上的方法
4. 利用计算出的信息构造一个最优解

矩阵链乘法 (续)

1. 刻画一个最优解的结构特征——最优括号化方案的结构特征

   - $A_{i..j}$ $(1 \le i \le j \le n)$：$A_i A_{i+1} \cdots A_j$
   - 设$A_i A_{i+1} \ldots A_j$的最优括号化是在A_k和A_{k+1}之间划分开 $(i \le k < j$且$i < j)$
   - 对某个k，先计算$A_{i..k}$和$A_{k+1..j}$，再计算两者乘积得到$A_{i..j}$
   - 计算代价：

     $A_{i..k}$计算代价 + $A_{k+1..j}$计算代价 + 两者乘积计算代价

矩阵链乘法 (续)

- 关键： $A_iA_{i+1}\cdots A_j$的最优括号化亦要求分割开的两个子链$A_{i..k}$和$A_{k+1..j}$是最优括号化。
- 可用反证法证明：若$A_{i..k}$括号化不是最优，则可找到一个成本更小的方法将其括号化，代入到$A_{i..j}$的最优括号化表示中，得到的计算成本比最优解小，矛盾！

矩阵链乘法 (续)

2. 递归地定义最优解的值

- 怎样用子问题的最优解递归地定义原问题的最优解(一般是最优解的值)?
- 子问题：对所有的$1 \le i \le j \le n$确定$A_{i..j}$最优括号化代价
- $m[i, j]$：计算$A_{i..j}$所需标量乘法次数的最小值
  - 若$i=j$，矩阵链只有A_i，无需乘法，$m[i, i]=0$
  - 若$i<j$，利用步骤1最优子结构计算代价（k是最优分割点）：
    $A_{i..k}$计算代价 + $A_{k+1..j}$计算代价 + 两者乘积计算代价
    即：$m[i, j] = m[i, k] + m[k+1, j] + p_{i-1}p_k p_j$
- 原问题最优解：计算$A_{1..n}$所需的最小代价为$m[1, n]$

矩阵链乘法 (续)

■$m[i, j]$：计算$A_{i..j}$所需标量乘法次数的最小值

- 若$i<j$，利用步骤1最优子结构计算代价（k是最优分割点）：
  $A_{i..k}$计算代价 + $A_{k+1..j}$计算代价 + 两者乘积计算代价
  即：$m[i, j] = m[i, k] + m[k+1, j] + p_{i-1}p_k p_j$
- 以上公式成立需要k是最优分割点
- 检查所有$j-i$种可能的k即可：

$$m[i,j] = \begin{cases} 0, & i = j, \\ \min\limits_{i \le k < j} \{m[i,k] + m[k+1,j] + p_{i-1}p_k p_j\}, & i < j. \end{cases}$$

- 定义$s[i, j]$保存$A_{i..j}$最优括号化方案分割点位置k

矩阵链乘法 (续)

■例：求矩阵链<A_1, A_2, A_3, A_4, A_5>乘积，在k=3处分割：

➢$(A_1\ A_2\ A_3\ A_4\ A_5)$

$$(A_1 \quad A_2 \quad A_3) \qquad (A_4 \quad A_5)$$

$$p_0 \times p_1 \quad p_1 \times p_2 \quad p_2 \times p_3 \qquad p_3 \times p_4 \quad p_4 \times p_5$$

$$\Downarrow \qquad\qquad \Downarrow$$

$$p_0 \times p_3 \qquad\qquad p_3 \times p_5$$

➢标量乘法次数：$m[1, 5] = m[1, 3] + m[4, 5] + p_0 p_3 p_5$

$$m[i,\ j] = m[i,\ k] + m[k+1, j] + p_{i-1} p_k p_j$$

矩阵链乘法 (续)

3. 计算最优解的值，通常采用自底向上的方法

$$m[i,j]=\begin{cases}0, & i=j,\\ \min\limits_{i\le k<j}\{m[i,k]+m[k+1,j]+p_{i-1}p_kp_j\}, & i<j.\end{cases}$$

- 递归算法为指数时间复杂度：递归调用树不同分支中多次计算同一个子问题
- 需求解的不同子问题数目为n^2阶：$C_n^2+n=\Theta(n^2)$
  每对满足$1\le i\le j\le n$的i和j对应一个唯一的子问题
- 矩阵A_i规模为$p_{i-1}\times p_i$ (i=1,2,…,n)，辅助表$m[1..n, 1..n]$保存代价$m[i, j]$，辅助表$s[1..n\text{-}1, 2..n]$记录$m[i, j]$对应的最优分割点k

矩阵链乘法 (续)

■例：计算矩阵链<A_1, A_2, A_3, A_4>乘积。以下计算所有$m[i, j]$的值，其中$1 \le i \le j \le n$

- $i = j$：$m[1, 1] = m[2, 2] = m[3, 3] = m[4, 4] = 0$
- $i < j$：$m[1, 2]$：k=1时为$m[1, 1] + m[2, 2] + p_0p_1p_2$

  $m[2, 3]$：k=2时为$m[2, 2] + m[3, 3] + p_1p_2p_3$

  $m[3, 4]$：k=3时为$m[3, 3] + m[4, 4] + p_2p_3p_4$

  $m[1, 3]$：k=1时为$m[1, 1] + m[2, 3] + p_0p_1p_3$

  k=2时为$m[1, 2] + m[3, 3] + p_0p_2p_3$

  $m[2, 4]$：k=2时为$m[2, 2] + m[3, 4] + p_1p_2p_4$

  k=3时为$m[2, 3] + m[4, 4] + p_1p_3p_4$

矩阵链乘法 (续)

■例：计算矩阵链<A_1, A_2, A_3, A_4>乘积。以下计算所有$m[i, j]$的值，其中$1 \le i \le j \le n$

- $i = j$：$m[1, 1] = m[2, 2] = m[3, 3] = m[4, 4] = 0$
- $i < j$：$m[1, 2], m[2, 3], m[3, 4]$

  $m[1, 3], m[2, 4]$

  $m[1, 4]$：k=1时为$m[1, 1] + m[2, 4] + p_0p_1p_4$

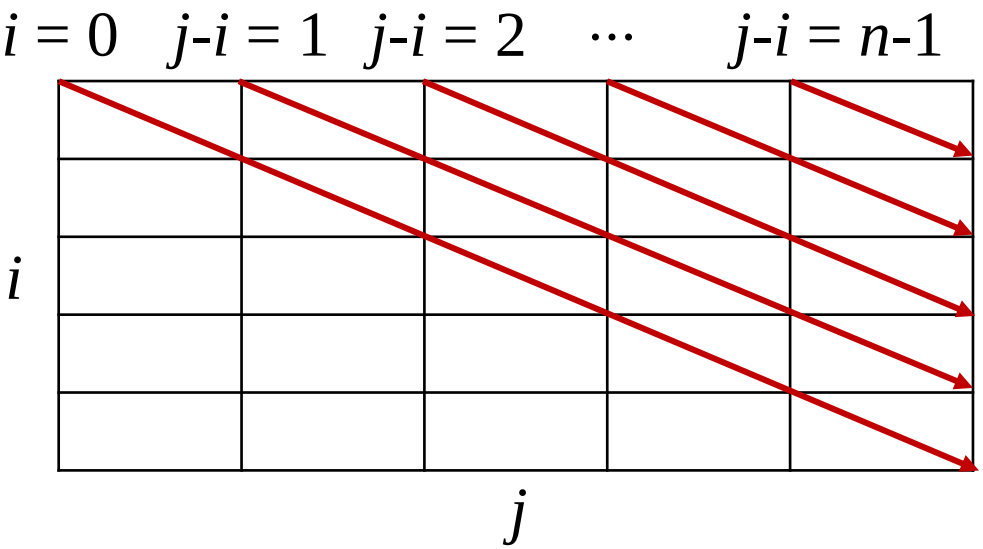
  k=2时为$m[1, 2] + m[3, 4] + p_0p_2p_4$

  k=3时为$m[1, 3] + m[4, 4] + p_0p_3p_4$

矩阵链乘法 (续)

■$m[i, j]$ 计算顺序：

$$j\text{-}i = 0,\ j\text{-}i = 1,\ j\text{-}i = 2,\ \ldots,\ j\text{-}i = n\text{-}1$$



矩阵链乘法 (续)

```
MATRIX_CHAIN_ORDER(p)
1   n ← p.length - 1
2   let m[1..n, 1..n] and s[1..n-1, 2..n] be new tables
3   for i ← 1 to n do
4       m[i, i] ← 0    // i = j
5   for l ← 2 to n do  // l：矩阵链长度，i ≠ j 时 1 ≤ j-i = l-1 ≤ n-1
6       for i ← 1 to n-l+1 do
7           j ← i + l - 1
8           m[i, j] ← ∞
9           for k ← i to j-1 do
10              q ← m[i, k] + m[k+1, j] + p_{i-1} p_k p_j
11              if q < m[i, j]
12                  m[i, j] ← q
13                  s[i, j] ← k
14  return m and s
```

矩阵链乘法 (续)

- 时间复杂度：$O(n^3)$，三层循环
- 空间复杂度：$O(n^2)$，保存表m和s
- 较穷举方法指数阶高效得多

矩阵链乘法 (续)

4. 构造最优解

- MATRIX_CHAIN_ORDER求出了计算矩阵链乘积所需的最少标量乘法次数，但并未指出如何进行这种最优代价矩阵链乘法计算
- 表s记录了构造最优解的最优分割信息，可递归求出其中最外层划分位置：$k = s[1, n]$，则进一步求$s[1, k]$和$s[k+1, n]$，直到$s[i, j]$中$i=j$为止

```
PRINT_OPTIMAL_PARENS(s, i, j)
1  if i = j
2      print "A"_i
3  else print "("
4      PRINT_OPTIMAL_PARENS(s, i, s[i, j])
5      PRINT_OPTIMAL_PARENS(s, s[i, j]+1, j)
6      print ")"
```

矩阵链乘法 (续)

■按照最优括号化计算矩阵链乘积

```
MATRIX_CHAIN_MULTIPLY(A, s, i, j)
1  if i = j
2      return A_i
3  else
4      X ← MATRIX_CHAIN_MULTIPLY(A, s, i, s[i, j])
5      Y ← MATRIX_CHAIN_MULTIPLY(A, s, s[i, j]+1, j)
6      return MATRIX_MULTIPLY(X, Y)
```

动态规划问题

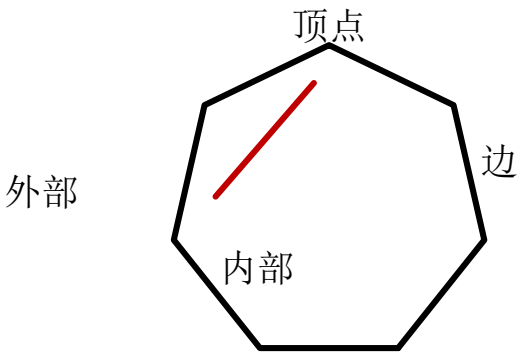
- 钢条切割
- 矩阵链乘法的最优括号化
- 多边形的最佳三角剖分
- 最长公共子序列
- 最优二叉搜索树
- 0-1背包

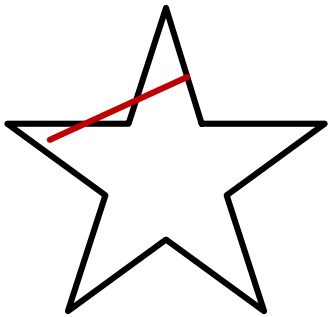
多边形的最佳三角剖分

■凸多边形的最佳三角剖分类似于矩阵链乘问题

■凸多边形

- 多边形：无交叉边
- 凸多边形：边界上或内部的任意两点连线上的点均在边界上或内部





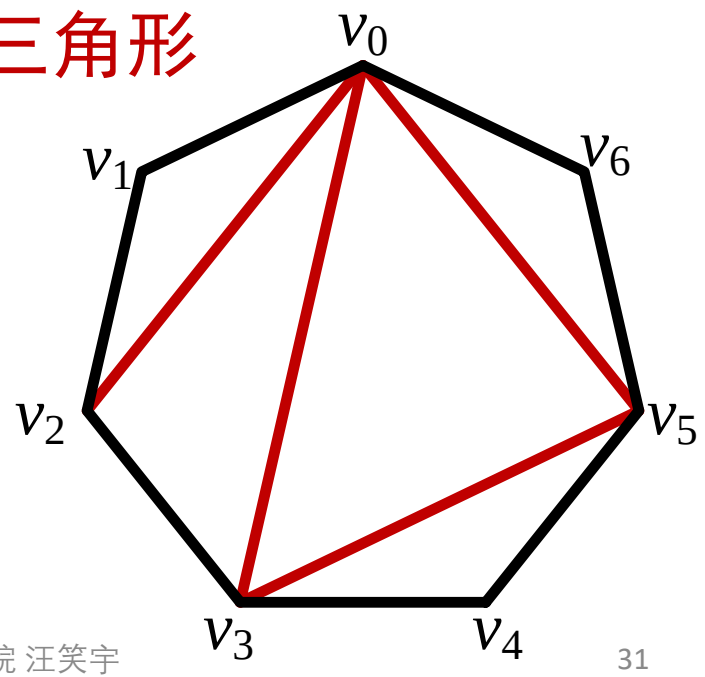
■表示

- 顶点逆时针列表：$P_n=<v_0, v_1, \ldots, v_{n-1}>$
- n条边：$\overline{v_0v_1}, \overline{v_1v_2}, \ldots, \overline{v_{n-1}v_n}, \quad v_0 = v_n$
- 顶点号是顶点数模除结果

以下均讨论凸多边形！

多边形的最佳三角剖分 (续)

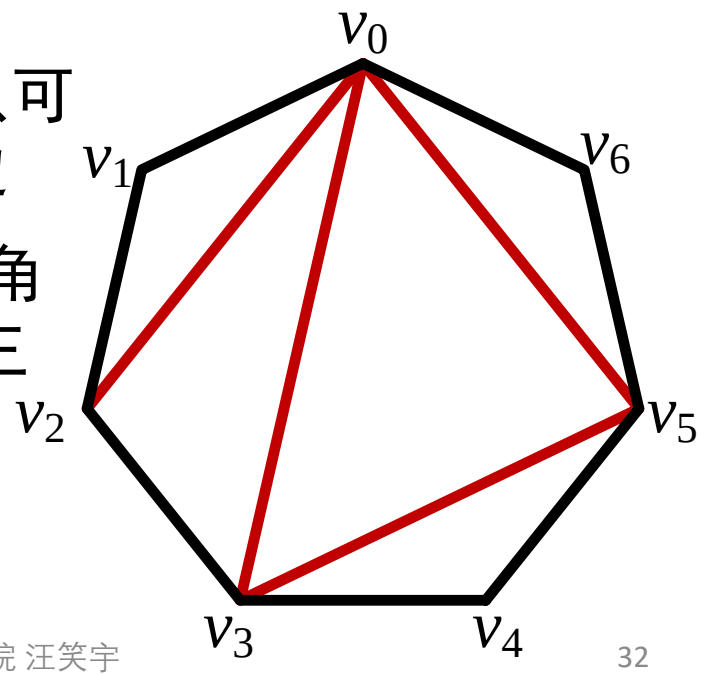
- 弦：若v_i , v_j是不相邻点，则线段 $\overline{v_i v_j}$ 是一条弦，它将多边形划分为两个多边形：
  $<v_i, v_{i+1}, \ldots, v_j>$ 和 $<v_j, v_{j+1}, \ldots, v_i>$
- 多边形的三角剖分：多边形的一个弦集T，将多边形划分为若干个不相交的三角形



多边形的最佳三角剖分 (续)

■一些结论：

- 在一个三角剖分中，没有弦在除端点外的位置相交且弦集合T最大（每条不在T中的弦与T中的某弦相交于端点外的位置）
- 三角剖分产生的三角形的边只可能是剖分中的弦或多边形的边
- n个顶点的凸多边形的每个三角剖分有n-3条弦，划分所得的三角形个数为n-2



多边形的最佳三角剖分 (续)

■最优的三角剖分
给定多边形及三角形的权值函数W，找到权值之和最小的三角剖分

- 三角形权值：视具体问题而定，例如：

$$W(\Delta v_i v_j v_k) = |v_i v_j| + |v_j v_k| + |v_k v_i|$$

$|v_i v_j|$是v_i和v_j的欧氏距离

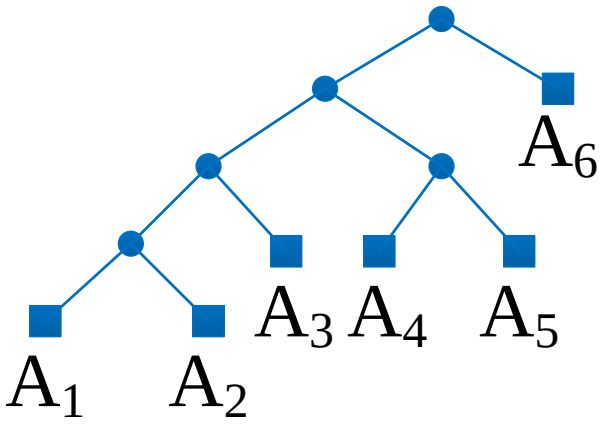
- 三角剖分的权：各三角形权值之和
- 最优三角剖分：权值之和最小的三角剖分

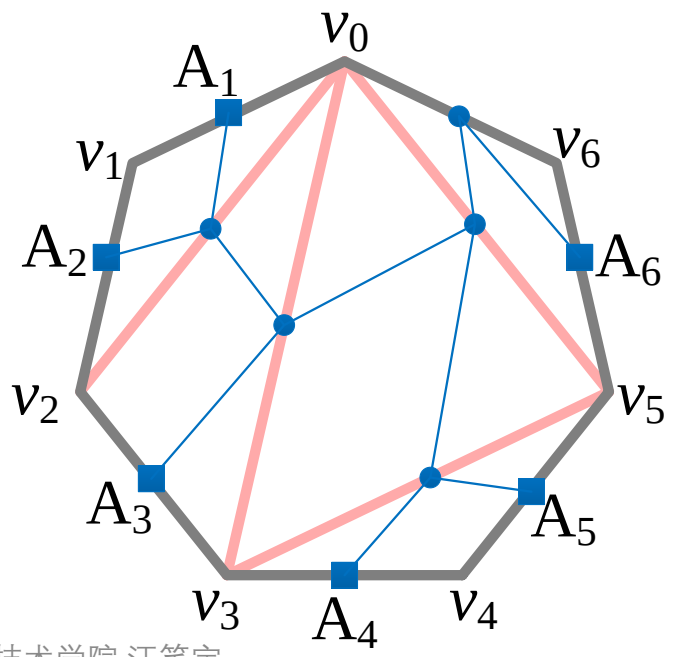
多边形的最佳三角剖分 (续)

■与矩阵链乘法的括号化一致

通过树来解释三角剖分与表达式括号化的一致性

$(((A_1A_2)A_3)(A_4A_5))(A_6)$





多边形的最佳三角剖分 (续)

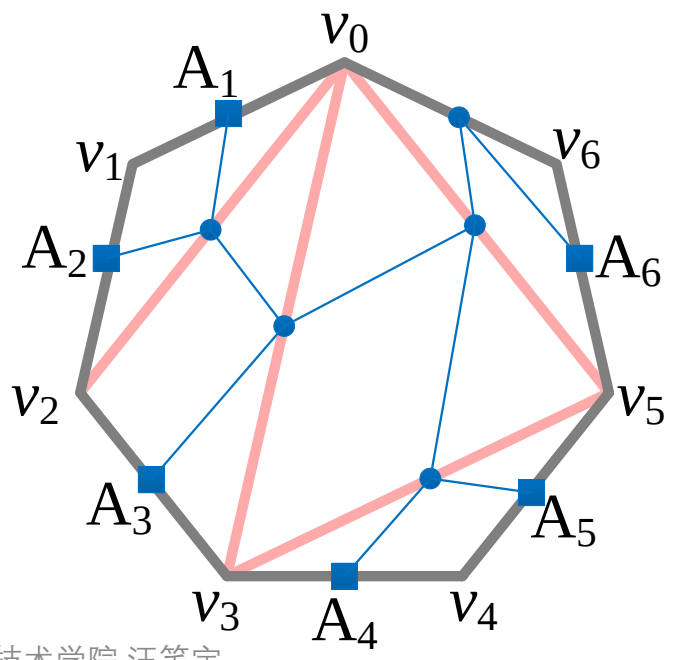
■n个矩阵的完全括号化

⇔ n个叶子的解析树 (parse tree)

⇔ n+1个顶点的多边形的三角剖分

➢矩阵A_i (大小为$p_{i-1} \times p_i$)对应边 $\overline{v_{i-1}v_i}$ (i=1, 2, ..., n-1)

➢矩阵积$A_{i+1..j}$对应弦 $\overline{v_i v_j}$



多边形的最佳三角剖分 (续)

- 最优矩阵链乘可看作是最优三角剖分的特例：
  - 定义三角剖分的权函数：$W(\Delta v_i v_j v_k) = p_i p_j p_k$
  - 该权函数下的P_{n+1}的最优三角剖分就给出了$A_1 A_2 \cdots A_n$的一个最优括号化
- 求解最优三角剖分问题
  - 用$v_0, v_1, \ldots, v_n$代替$p_0, p_1, \ldots, p_n$
  - 修改MATRIX_CHAIN_ORDER中q计算公式为：
    $q \leftarrow m[i, k] + m[k+1, j] + W(\triangle v_{i-1} v_k v_j)$

$m[1, n]$包含了P_{n+1}的一个最优三角剖分的权值，为什么？

n个矩阵进行链乘：n-1次标准矩阵乘法；n+1边形三角剖分：划分出n-1个三角形

多边形的最佳三角剖分 (续)

1. 刻画一个最优解的结构特征——最优三角剖分的子结构

   - 设多边形$P_{n+1} = <v_0, v_1, ..., v_n>$的一个最佳三角剖分包括某个三角形$\triangle v_0 v_k v_n$，则权值

$$W(P_{n+1}) = W(\triangle v_0 v_k v_n) + W(<v_0, v_1, ..., v_k>) + W(<v_k, v_{k+1}, ..., v_n>)$$

   - $W(P_{n+1})$最优就要求子多边形$<v_0, v_1, ..., v_k>$和$<v_k, v_{k+1}, ..., v_n>$的三角剖分也是最优的

多边形的最佳三角剖分 (续)

2. 递归地定义最优解的值

- 设$t[i, j](1 \le i \le j \le n)$是多边形$<v_{i-1}, v_i, \ldots, v_j>$的一个最优解的值（即最优三角剖分的权），则多边形$P_{n+1}$的最优三角剖分的权为$t[1, n]$。
- $t[i, j]$的递归定义如下：
  - 若$i=j$，多边形退化为$<v_{i-1}, v_i>$，可定义权为0 即$t[i, i]=0, 1 \le i \le n$
  - 若$i<j$, 则多边形$<v_{i-1}, v_i, \ldots, v_j>$至少有3个顶点，设最佳剖分点为$k$ $(i \le k < j)$，则剖分结果为：
    $\triangle v_{i-1}v_kv_j$ $<v_{i-1}, v_i, \ldots, v_k>$ $<v_k, v_{k+1}, \ldots, v_j>$

多边形的最佳三角剖分 (续)

2. 递归地定义最优解的值

- $t[i, j]$的递归定义如下：

$$t[i,j]=\begin{cases}0, & i=j,\\ \min\limits_{i\le k<j}\{t[i,k]+t[k+1,j]+W(\Delta v_{i-1}v_kv_j)\}, & i<j.\end{cases}$$

3. 计算最优解的值 & 4. 构造最优解

- 与矩阵链完全一致，仅权函数不同
- 时间复杂度：$O(n^3)$；空间复杂度：$O(n^2)$

动态规划原理

- 最优子结构
- 注意事项
- 重叠子问题
- 重构最优解
- 备忘

最优子结构

- 最优子结构性质：一个问题的最优解包含其子问题的最优解
  - 例：矩阵$A_iA_{i+1}\cdots A_j$的最优括号化问题蕴含着两个子问题$A_i\cdots A_k$和$A_{k+1}\cdots A_j$的解也必须是最优的
- 具有最优子结构性质的问题可能会使用动态规划
- 在动态规划中，可用子问题的最优解来构造原问题的最优解

最优子结构 (续)

- 如何发现最优子结构？
  - 说明问题的解必须进行某种选择，这种选择导致一个或多个待解的子问题
  - 对一给定问题，假定导致最优解的选择已给定，即无须关心如何做出选择，只须假定它已给出
  - 给定选择后，决定由此产生哪些子问题，如何最好地描述子问题空间
  - 证明用在问题最优解内的子问题的解也必须是最优的。方法是“剪切-粘贴”(cut-and-paste)技术和反证法。假定在最优解对应的子问题的解非最优，删去它换上最优解，得到原问题的解非最优，矛盾！

最优子结构 (续)

- 如何描述子问题空间（不同的子问题个数）：尽可能使其简单，然后再考虑有没有必要扩展
  - 钢条切割：对每个i值，长度为i钢条的最优切割
  - 矩阵链乘法：$A_{i..j}$
- 最优子结构有关的两方面问题
  - 原问题最优解中涉及多少个子问题
  - 用在最优解中的子问题有多少种选择
    - 钢条切割：一个子问题，n种选择

$$r_n = \max_{1 \le i \le n} (p_i + r_{n-i})$$

    - 矩阵链乘法：两个子问题，j-i种选择

$$m[i,j] = \begin{cases} 0, & i = j, \\ \min\limits_{i \le k < j} \{m[i,k] + m[k+1,j] + p_{i-1} p_k p_j\}, & i < j. \end{cases}$$

最优子结构 (续)

- 动态规划算法的运行时间
  - 子问题总数
  - 对每个子问题涉及多少种选择
- 例：矩阵链乘共要解$\Theta(n^2)$个子问题: $1 \le i \le j \le n$，求解每个子问题至多有n-1种选择，最终的运行时间为$\Theta(n^3)$
- 动态规划求解方式
  - 自底向上

注意事项

- 注意问题是否具有最优子结构性质！
- 例：给定有向无权图$G=(V, E)$和顶点$u, v \in V$，求顶点u到v的最短/最长路径的问题（指简单路径）
  - 无权最短路径问题具有最优子结构性质
    设从u到v的最短路径是P，并设中间点为w，则

$$u \xrightarrow{P} v \quad \Rightarrow \quad u \xrightarrow{P_1} w \xrightarrow{P_2} v$$

显然P_1和P_2也必须是最优(短)的

注意事项 (续)

- 最长路径不具有最优子结构

  设P是从u到v的最长路径，w是中间某点，则

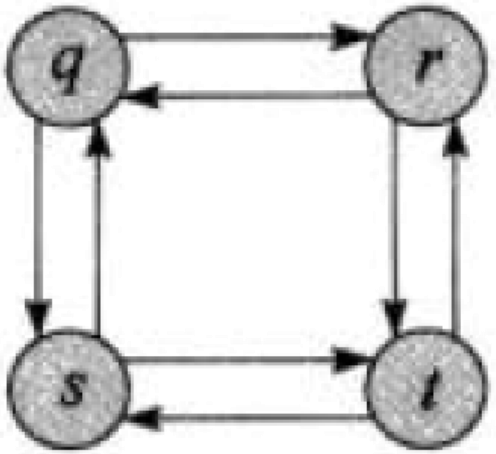
$$u \xrightarrow{P} v \quad \Rightarrow \quad u \xrightarrow{P_1} w \xrightarrow{P_2} v$$

  但P_1不一定是从u到w的最长路径，P_2也不一定是从w到v的最长路径

- 例：考虑一条最长路径：$q \to r \to t$

  但q到r的最长路径是：$q \to s \to t \to r$

  r到t的最长路径是：$r \to q \to s \to t$



注意事项 (续)

- 为什么两问题有差别？
  - 最长路径的子问题不是独立的
    所谓独立指一个子问题的解不能影响另一个子问题的解
    但第一个子问题中使用了s和t，第二个子问题又使用了，使得产生的路径不再是简单路径
    从另一个角度看：一个子问题求解时使用的资源(顶点)不能在另一个子问题中再使用
  - 最短路径问题中，两子问题没有共享资源，可用反证法证明
- 再如矩阵链乘法：$A_{i..j} \Rightarrow A_{i..k} \cdot A_{k+1..j}$
  显然两子链不相交，是相互独立的两子问题

重叠子问题

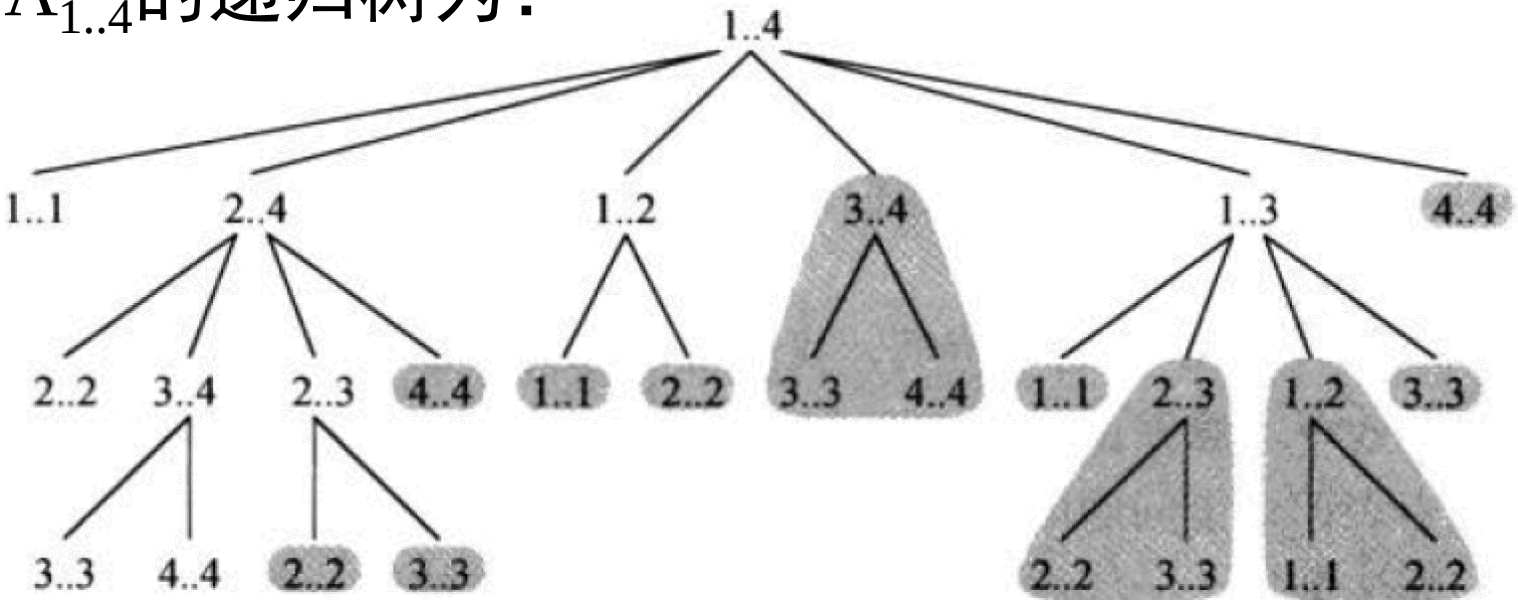
- 子问题空间足够小：用递归算法解某问题时重复计算同一子问题
- 分治法与动态规划的比较
  - 当递归每一步产生一个新的子问题时，适合使用分治法
  - 当递归中较多出现重叠子问题时，适合使用动态规划，即对重叠子问题只求解一次，然后存储在表中，当需要使用时常数时间内查表
  - 若子问题规模是多项式阶的，动态规划特别有效

重叠子问题 (续)

■例：用自然递归算法求解矩阵链乘法（算法见教材p219）

$$m[i,j]=\begin{cases}0, & i=j,\\ \min\limits_{i\le k<j}\{m[i,k]+m[k+1,j]+p_{i-1}p_kp_j\}, & i<j.\end{cases}$$

$A_{1..4}$的递归树为：



重叠子问题 (续)

■阴影部分是重叠子问题，递归算法须重复计算

$$m[1,n]=\begin{cases}0, & n=1,\\ \min\limits_{1\le k<n}\{m[1,k]+m[k+1,n]+p_0p_kp_n\}, & n>1.\end{cases}$$

■递归式时间：$\begin{cases}T(1)\ge 1,\\ T(n)\ge 1+\sum\limits_{k=1}^{n-1}(T(k)+T(n-k)+1), & n>1.\end{cases}$

代入法得$T(n)=\Omega(2^n)$

■结论：当自然递归算法是指数阶，但实际不同的子问题数目是多项式阶时，可用动态规划来获得高效算法

重构最优解

- 用附加表保存中间选择结果能节省重构最优解的时间
  - 例：矩阵链乘法利用表s记录最优分割位置